METECS
Engineering & Applied Technologies

P.O. Box 891506
Houston, TX

mREST

Interface Specification

METECS-R-XXX
Revision: DRAFT
Date: 5/31/2012

Prepared By: _____

Patrick McCartney, P.E. – mREST Project Manager

Reviewed By: _____

John MacLean, P.E. – mREST Chief Engineer

Document Change Log

Date	Revision	Section(s)	Description of Change
5/31/12	Baseline	All	Initial document release

Table of Contents

1 INTRODUCTION 1

1.1 BACKGROUND AND OVERVIEW 1
1.1.1 REPRESENTATIONAL STATE TRANSFER (REST) 1
1.1.2 MREST 4
1.2 PURPOSE AND SCOPE 6
1.3 APPLICATION 7
1.4 OUTLINE 7

2 REFERENCES 8

3 DEFINITIONS, ACRONYMS, AND ABBREVIATIONS 9

3.1 DEFINITIONS 9
3.2 ACRONYMS AND ABBREVIATIONS 11

4 ARCHITECTURAL APPROACH 13

4.1 KEY TECHNOLOGIES 13
4.1.1 OPEN STANDARDS 13
4.1.2 REST WEB SERVICES 15
4.1.3 DISCOVERY 16
4.1.4 EXTENSIBLE MARKUP LANGUAGE (XML) 16
4.1.5 AUTOMATIC TEST MARKUP LANGUAGE (ATML) 17
4.1.6 OPEN SOURCE 17
4.2 SYSTEM COMPONENTS 18
4.2.1 LOGICAL SYSTEM ELEMENT (LSE) 18
4.2.2 MREST MANAGERS 19
4.3 INTERNAL LSE DATA MODELING 20
4.3.1 PARAMETERS AND RESOURCES 20
4.3.2 STATIC AND DYNAMIC PARAMETER DATA 21

5 MREST INTERFACE DEFINITION 23

5.1 RESOURCE TYPES 23
5.1.1 DOCUMENT RESOURCES 23
5.1.2 COMPONENT RESOURCES 24
5.1.3 COLLECTION RESOURCES 24
5.1.4 CATALOG RESOURCES 25
5.1.5 CONTROLLER RESOURCES 25
5.1.6 DATALOG RESOURCES 27
5.2 STANDARDIZED CONTROLLER RESOURCES 28
5.2.1 DATA LOGGING (/MREST/CONTROLLERS/LOGDATA) 28
5.3 MREST UNIFORM INTERFACE 31
5.3.1 HTTP REQUESTS 31
5.3.2 HTTP RESPONSES 31
5.3.3 HTTP HEADERS 31
5.4 MREST SERVER RESOURCES 31
5.4.1 IDENTIFICATION 31
5.4.2 DATA LOGGING 31
5.4.3 APPLICATION SPECIFIC RESOURCES 31
5.5 MREST RESOURCE REPRESENTATIONS 32
5.5.1 RESOURCE REPRESENTATION FORMATS 32
5.5.2 MEDIA TYPE NEGOTIATION 32
5.6 MREST STATUS REQUESTS 33
5.7 MREST CONFIGURATION UPDATE REQUESTS 33
5.8 MREST LOGICAL SYSTEM ELEMENT DISCOVERY 33
5.9 MREST SECURITY 33

6 MREST MANAGER REQUIREMENTS 34

6.1 LSE INTERFACE 34
6.1.1 UNIFORM INTERFACE 34
6.1.2 LOCATIONS OF LSE RESOURCES 34
6.1.3 DATA REPRESENTATIONS OF RESOURCES 34
6.1.4 CACHING 34
6.1.5 XML VALIDATION 34
6.1.6 STATELESS COMMUNICATION 34

6.1.7 CALLBACK 34
6.1.8 OPTIMIZATION 34
6.2 OPERATOR INTERFACE 34
6.3 INITIALIZATION OF THE SYSTEM AND SESSION CONFIGURATION 34
6.4 COORDINATION OF SESSION RUN EXECUTION 34
6.5 COLLECTION AND DISPLAY OF STATUS DATA 35
6.5.1 DATABASE VERIFICATION 35
6.5.2 STATUS LOG REQUEST DATA 35
6.6 COLLECTION OF SESSION MANAGER NOTES 35
6.7 ORGANIZATION OF SYSTEM SESSION RESULTS 35

7 EXAMPLE IMPLEMENTATIONS 36

7.1 TEST FLOW DATA MANAGER (TFDM) 36
7.2 LOGICAL TEST ELEMENT (LTE) 36
7.3 STANDALONE TEST EXECUTIVE (STX) 36

APPENDIX A MREST REQUIREMENTS MATRIX 37

APPENDIX B MREST RESOURCE REPRESENTATIONS 38

B.1 MREST XML (APPLICATION/VND.MREST+XML) 38
B.2 MREST JSON (APPLICATION/VND.MREST+JSON) 38
B.3 ATML (APPLICATION/VND.ATML+XML) 38

APPENDIX C MREST RESOURCE TYPES 39

C.1 MREST XML (APPLICATION/VND.MREST+XML) 39
C.2 MREST JSON (APPLICATION/VND.MREST+JSON) 39
C.3 ATML (APPLICATION/VND.ATML+XML) 39

List of Tables

Table 1: Sample Resource at http://www.mrest.org/mrest/info 2
Table 2: Resource Representations from http://www.mrest.org/rest/info 3
Table 3: Static and Dynamic Parameter Data 21
Table 4: Sample LSE Parameter 22
Table 3: mREST Resource Types 23
Table 4: Methods for mREST Document Resources 23
Table 5: Methods for mREST Component Resources 24
Table 6: Methods for mREST Collection Resources 24
Table 7: Methods for mREST Catalog Resources 25
Table 8: Methods for mREST Controller Resources 27
Table 9: Methods for mREST Controller Exec, Input, Status, and PID Resources 27
Table 10: Methods for mREST Datalog Resources 27
Table 11: Inputs for Datalog Controller Resource 29
Table 12: Sample XML Input for Datalog Controllers 30
Table 12: XML Schema for Datalog Controller Inputs 30
Table 10: mREST Requirements Matrix 37
Table 11: mREST Resource Types 40

List of Figures

Figure 1: The REST Architectural Style 1
Figure 2: Network Interaction for a Simple Internet Search 2
Figure 3: mREST System Overview 4
Figure 4: Example of an mREST System 5
Figure 5: mREST Interface Specification Document Scope 6
Figure 6: Logical System Element (LSE) Architecture 18
Figure 7: Controller Status Resources 26

1 Introduction

1.1 Background and Overview

1.1.1 Representational State Transfer (REST)

REST, or Representational State Transfer, is a term originally used by Roy Fielding in his Ph.D. dissertation[1] to describe an architectural style for networked systems. The REST architectural style provides a philosophy for identifying items of interest in a system as *resources*, modeling those resources using a known format (*representations*) which are referenced through the use of a *uniform resource identifier (URI)*, and then specifying a *uniform method* for retrieving and/or manipulating those resources. An overview of the REST architectural style is shown in Figure 1.

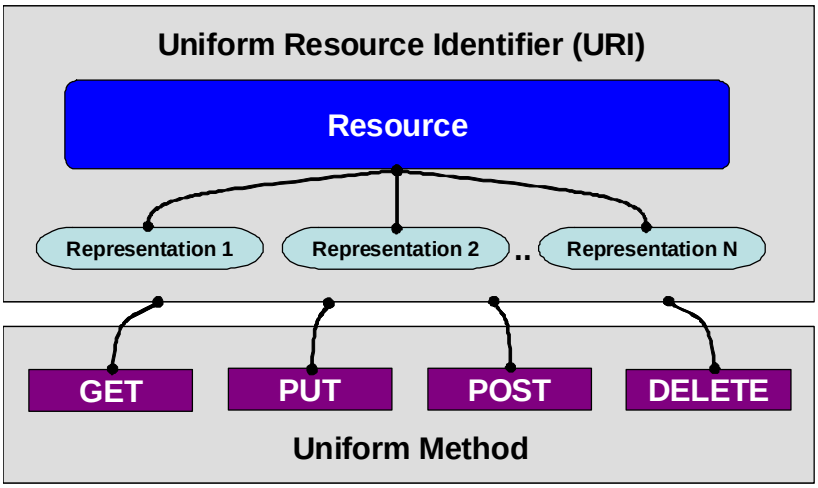


Figure 1: The REST Architectural Style

A familiar implementation of a REST architecture is the world-wide-web (Figure 2). A user desires to do an Internet search so they open up a browser and enter the URI for a search engine such as http://www.google.com. The browser sends a GET command to the server at the URI which responds with a web page represented as text formatted using the Hyper Text Markup Language (HTML). The user then enters some search text into the appropriate field which the browser sends using a POST command. The server then responds with an HTML representation of a web page containing the search results.

For the network interaction shown in Figure 2, the URI is the web address of the Google server (http://www.google.com). The browser sends either GET or POST requests to the server depending on user input. The server responds by delivering the appropriate resource representations formatted as HTML text. The browser interprets the HTML text and displays a page that the user can easily understand.

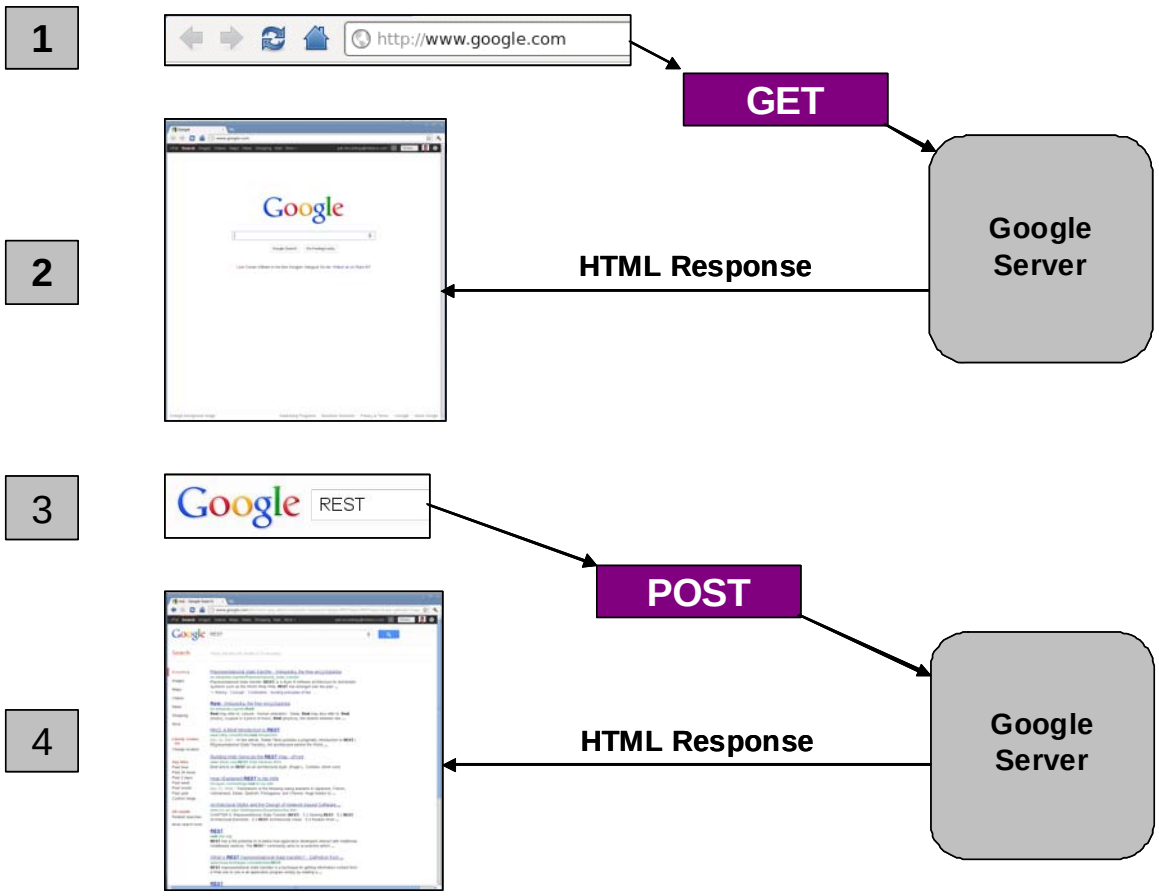


Figure 2: Network Interaction for a Simple Internet Search

As a more generic example of a network interaction using the REST architectural style, consider a resource at the URI http://www.mrest.org/mrest/info which contains the information shown in Table 1.

Table 1: Sample Resource at http://www.mrest.org/mrest/info

Parameter	Value
UUID	**821fede88cd14ad182d386255e661991**
Name	**mREST**
Type	**Apache PHP**
Version	**v1.36**

The resource provider decides to offer two different representations of the resource. The first representation is formatted using the eXtensible Markup Language (XML) and the second uses JavaScript Object Notation (JSON) as shown in Table 2.

Table 2: Resource Representations from http://www.mrest.org/rest/info

Format	Respresentation
XML	<?xml version="1.0"?> <info> <atom:link rel="self" href="http://www.mrest.org/rest/info"/> <uuid>821fede8-8cd1-4ad1-82d3-86255e661991</uuid> <name>mREST</name> <type>REST Architecture</type> <version>v1.36</version> </info>
JSON	{ "info": { "link" : { "rel" : "self", "href" : "http://www.mrest.org/rest/info" } "uuid" : "821fede8-8cd1-4ad1-82d3-86255e661991", "name" : "mREST", "type" : "REST Architecture", "version" : "v1.36" } }

Any client is free to retrieve the resource using a GET method call to the URI. If the client requests the resource in XML format then the server will respond with the representation shown in the first row of Table 2. Likewise, if the client requests the resource in JSON format, the server will respond with the representation shown in the second row of Table 2.

Similarly, if the client wishes to update the resource (and the resource was configured to allow updates from a client), it could use the PUT method to send an updated XML or JSON document to the URI. A subsequent GET request would result in the server sending a new representation of the resource which included the requested updates.

The client/server interaction described in the previous examples is typical of any software architecture using the REST style. Such an architecture is sometimes referred to as "*RESTful*". The specific representation formats and other details such as how resources are modeled and how each method is handled will be different depending on the implementation. Additional information on the REST architectural style can be found in references [2], [3], [4], and [5].

1.1.2 mREST

mREST is an implementation of the REST architecture specific to the management and sharing of data in a system of logical elements. A Logical System Element (LSE) can represent any number of varying types of hardware and/or software components or subsystems. As shown in Figure 3, each LSE acts as a server which communicates with one or more mREST managers (clients) using a protocol based on principles from the REST architectural style.

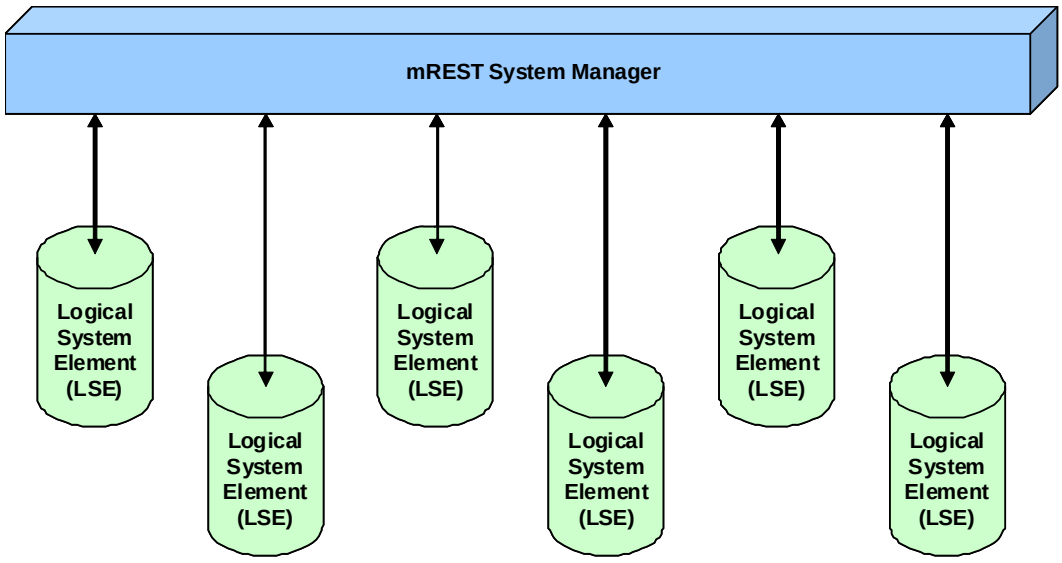


Figure 3: mREST System Overview

Along with the implementation of a *RESTful* application programming interface (API), mREST uses open-standards to handle additional tasks such as automatic discovery of LSE servers, how LSEs are identified and tracked in the system, and how data logging is handled in each system component. These additional specifications are what makes mREST useful for real-world applications. For example, the discovery specification makes it possible for LSEs to enter and/or exit the system without affecting the overall software architecture and without requiring interaction from the system manager. Similarly, a common specification for data logging provides a means for efficiently collecting data from the diverse collection of LSEs and for durable long-term archival of that data.

A sample instance of an mREST system is shown in Figure 4. This example demonstrates a configuration to support the testing of a motor for a robotic manipulator. Each of the hardware and software components are configured as separate LSEs. The mREST manager is able to discover which LSEs are available for the test and then orchestrate the test itself. Data is then collected from each LSE using the common mREST client/server interface. The mREST protocol is what allows such a diverse

arrangement of hardware and software to be managed as a single system in a coordinated manner.

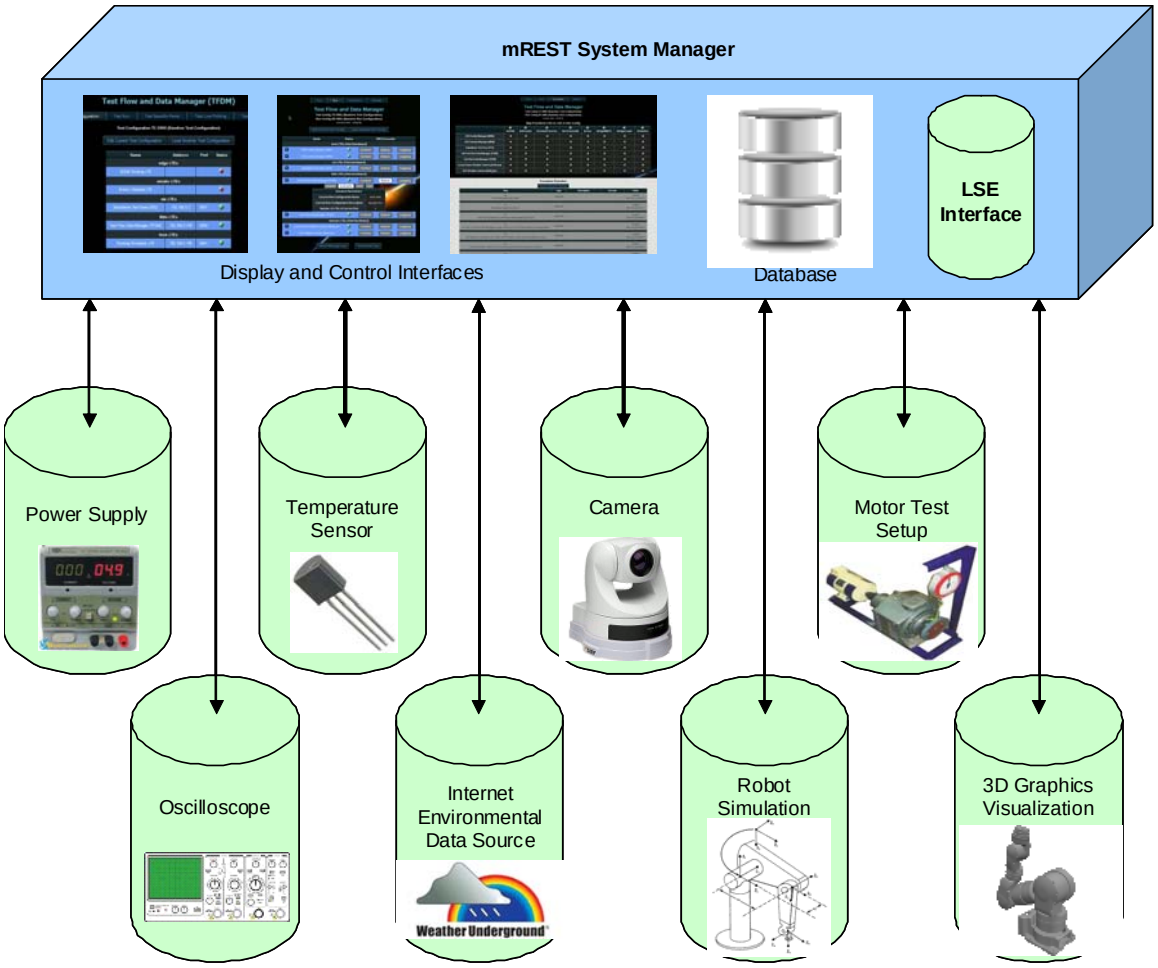


Figure 4: Example of an mREST System

Note that in this system even the mREST manager has an LSE interface. This allows the manager to communicate with itself using the same mREST protocol. It also provides a mechanism for several systems to be tiered and controlled by a higher-level manager. For example, the Robot Simulation LSE could actually be a separate subsystem comprised of an mREST manager and a series of simulation component LSEs. mREST provides the common framework that allows the system designer to model the system logically without sacrificing modularity, expandability, or reusability.

1.2 Purpose and Scope

The purpose of this document is to clearly define the mREST interface protocol. The interface protocol covers all of the interaction between mREST clients and mREST servers (Figure 5). System-level requirements are not specifically addressed.

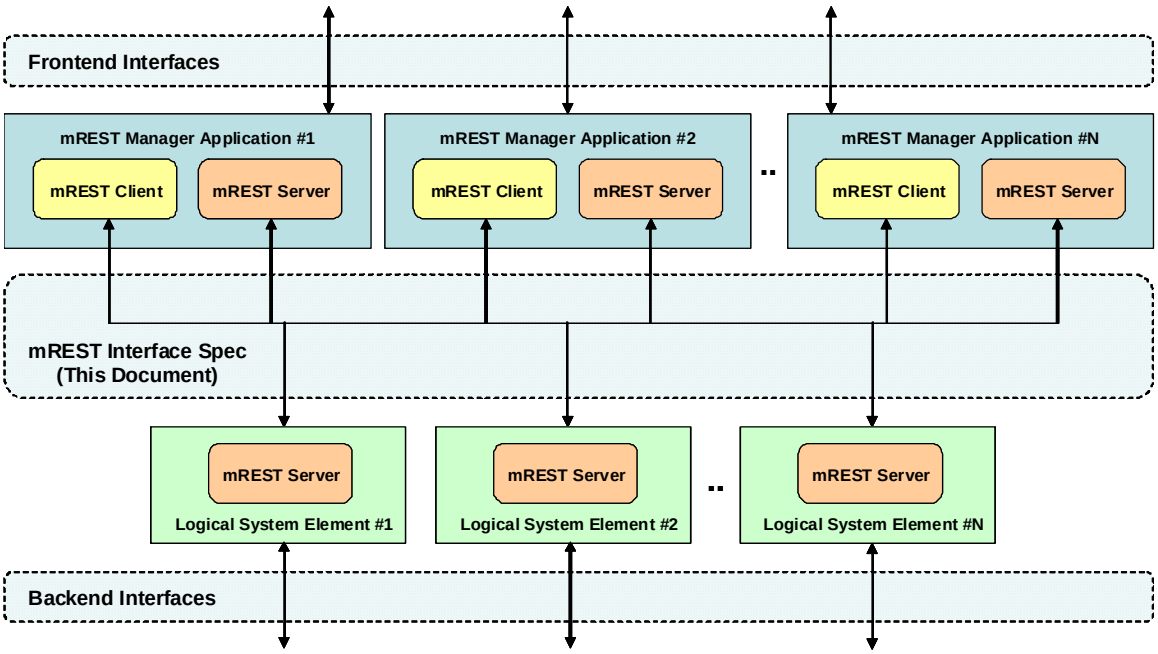


Figure 5: mREST Interface Specification Document Scope

In an mREST system, there are typically some "backend" interfaces between an LSE and the associated hardware/software system. For example, a network camera LSE would have a backend interface to the camera itself. These interfaces are specific to each type of LSE and are not covered in this document.

There are also "frontend" interfaces that may exist in certain mREST manager applications. For example, an electronic procedure execution application may have a specialized interface for configuring the procedures. This interface would be application specific and outside of this document scope.

mREST is intended to be a generic protocol which can be used in a wide variety of applications. A few scenarios are discussed to provide additional clarity but, in general, application-specific implementations of mREST are not specifically addressed.

In short, this document is intended to provide all of the information necessary for an application developer to create mREST interface agents. This includes both mREST clients (mREST manager applications) and mREST servers (logical system elements, or LSEs).

1.3 Application

It has already been stated that an mREST system employs the REST architectural style, which is really an architectural philosophy rather than a specific protocol. Just because an application is *RESTful* does not mean that it can be easily integrated with other *RESTful* applications. An application that uses mREST has made a conscious decision to go beyond REST and adhere to a specific architecture and interface protocol. This results in an mREST application that can be easily integrated with other mREST applications.

In order for a system to be a good candidate for mREST, it needs to be adaptable to a client-server architecture and be modeled as a group of system Logical System Elements (LSEs) as presented in the previous section. An mREST system also implies a definite operations flow where inputs are inserted into the system and data is gathered.

The primary benefit to using mREST is that it facilitates automating tasks that would be difficult and/or time consuming using manual methods. mRest also can greatly leverage the number of software application modules that a single user can effectively manage. It also provides a common interface for disparate system components while maintaining loose coupling and minimizing absolute requirements that exclude non-conforming software. These features are sometimes referred to as “orchestration”. Systems which can benefit from this type of automation are good candidates for mREST. A couple of the key implementations of mREST are in the areas of test automation and integrated simulation facility management.

1.4 Outline

The mREST architectural approach is discussed in Section 4 while the actual interface specification is defined in Section 5. Specific requirements on each mREST component are presented in Section 6. An mREST requirements summary, specific implementation details, and other highly detailed information are presented in the Appendices.

2 References

[1] Fielding, Roy Thomas. *Architectural Styles and the Design of Network-based Software Architectures*. Doctoral dissertation, University of California, Irvine, 2000 (http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm)

[2] Richardson, Leonard and Sam Ruby. *RESTful Web Services*. O'Reilly, May 2007

[3] Allamaraju, Subbu. *RESTful Web Services Cookbook*. O'Reilly, March 2010

[4] Webber, Jim, Savas Parastatidis, and Ian Robinson. *REST in Practice*. O'Reilly, September 2010

[5] Masse, Mark. *REST API Design Rulebook*. O'Reilly, October 2011

[6]

3 Definitions, Acronyms, and Abbreviations

3.1 Definitions

Logical System Element (LSE)

A logical representation of a real-world system component or components. An LSE can represent any number of varying types of hardware and/or software components or subsystems. LSEs are the "servers" in the mREST architecture and form the basic building blocks of an mREST system.

mREST

An implementation of a REST architecture specific to the management, operation, and sharing of data in a system of logical elements

mREST Manager (MRM)

An application which serves a specific display and/or control function in an mREST system and are typically the only component that contain mREST client interfaces.

Orchestration

The operation and control of the components in an mREST system.

Parameter

An abstraction of an internal piece of LSE data.

Resource

An singe piece of data in the mREST interface.

Representational State Transfer (REST)

A term which describes a specific architectural style for networked systems.

RESTful

An application which utilizes the REST architectural style.

Resource

The intended conceptual target of a hypertext reference; a conceptual mapping to a set of entities

Uniform Resource Identifier

A string of characters used to identify a resource (e.g., http://www.mrest.org/rest/sample_resource)

3.2 Acronyms and Abbreviations

API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
ATML	Automatic Test Markup Language
ATS	Automatic Test Systems
COTS	Commercial Off-The-Shelf
DNS	Domain Name Service
DoD	Department of Defense
GUI	Graphical User Interface
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
ID	Identifier
IP	Internet Protocol
IRIG	Inter-Range Instrumentation Group
JSON	JavaScript Object Notation
LSE	Logical System Element
mDNS	Multicast DNS
NTP	Network Time Protocol
NxTest	Next Generation Automatic Test Systems
POE	Post Once Exactly
PID	Process Identifier
RAID	Redundant Array of Inexpensive Disks
REST	Representational State Transfer
SOAP	Simple Object Access Protocol
UUID	Universal Unique Identifier
URI	Uniform Resource Identifier
VPN	Virtual Private Network
W3C	World Wide Web Consortium
XML	eXtensible Markup Language

XSL	eXtensible Stylesheet Language
XSLT	eXtensible Stylesheet Language Transformation

4 Architectural Approach

A primary objective of the mREST specification is that it be flexible while enforcing the constraints necessary to meet the requirements of system automation. The mREST protocol has been designed so that specific applications can take advantage of this flexibility without sacrificing interoperability with other mREST components or systems.

Another key objective is that mREST be simple to implement. The protocol is designed to provide basic capabilities while still accommodating more complicated interactions. A system designer can certainly implement a very complex system using mREST but the protocol should not require such complexity for a simple system.

Finally, the mREST protocol is intended to be an "open" standard. It is built using non-proprietary components with the goal of allowing anyone to implement the specification without worrying about licensing issues.

A brief discussion of the key technologies and a high-level overview of the mREST architectural components is presented in the following subsections.

4.1 Key Technologies

This section briefly describes and provides references for the technologies upon which the mREST protocol is based.

4.1.1 Open Standards

There are several significant advantages of an open standards based architecture (over ad-hoc or proprietary architectures) that tend to reduce costs, increase long-term performance, and reduce risk over a long-term program. These include:

Easier interoperability between components supplied by different vendors because open standards are already defined and accepted by a wider community

a. Greater choice of vendors for development and maintenance which reduces cost and technical risk by eliminating single contractor points of failure
b. Availability of compatible software, hardware, development tools and support options from industry and user communities (especially with the emergence of the open source movement)
c. Because they are supported by industry consensus, open standards are frequently machine, operating system, and language independent. This provides less reliance on specific equipment, languages, or operating systems that may go obsolete.
d. A path exists to upgrade architecture components/capabilities as standards evolve to avoid obsolescence

The architectural perspective taken is similar in nature to the Open Systems approach used by the Department of Defense (DoD) Automatic Test Systems (ATS) Directorate to

lower life cycle costs and improve systems performance through use of standards-based architectures[1].

With the mass production and proliferation of desktop computers, high speed Ethernet interfaces, and web-based communication technologies, has come an emergence of sophisticated messaging protocols, database technologies, and equipment discovery techniques. A number of non-profit consortiums and organizations such as the World Wide Web Consortium (W3C), the Internet Engineering Task Force (IETF) and the Institute of Electrical and Electronics Engineers (IEEE) have defined and are promoting the open specifications upon which these technologies are being based.

Government departments such as the Department of Defense have also directly sponsored development of open specifications, including in the automation and test area, through programs such as the Next Generation Automatic Test Systems (NxTest).

Specific open standards of significant interest to this project are:

a. DNS Service Discovery[2] and Multicast DNS[3] (mDNS) specifications for Zeroconf[4] networking from the IETF
b. Hypertext Transfer Protocol[5] (HTTP) from the IETF
c. Extensible Markup Language[6] (XML) from the W3C
d. XML Schema[7] from the W3C
e. JSON…
f. ATOM…
g. Automatic Test Markup Language[8] (ATML) from the IEEE

One potential risk of using open specifications compared to ad-hoc specifications would be if system components had to be continuously upgraded to conform to the latest release of any of the specifications. This risk is mitigated first by baselining the mREST protocol to reference a specific set of open specification releases. Secondly, risk is mitigated by specifically avoiding proprietary or non-standard protocols that may require tight coupling between system elements (such as installable drivers that have to match exact versions of a driver on another piece of equipment to function properly). Finally,

[1] http://www.acq.osd.mil/ats

[2] http://www.dns-sd.org

[3] http://www.multicastdns.org

[4] http://www.zeroconf.org

[5] http://tools.ietf.org/html/rfc2616

[6] http://www.w3.org/XML

[7] http://www.w3.org/XML/Schema

[8] http://grouper.ieee.org/groups/scc20/tii

backward compatibility is enabled by including specification version numbers in the protocol.

4.1.2 REST Web Services

Web services was quickly identified as an applicable technology because of its compatibility with XML, widespread user communities, language and operating system independence, and adoption within the test industry in standards such as Automatic Test Markup Language (ATML). Two types of web services were prototyped: The Simple Object Access Protocol (SOAP) and REST. REST was selected because it provides a simpler, more flexible, and more elegant interface. Transparency of operation is also enhanced with REST because states of the system component interfaces are readily visible using simple tools such as web browsers.

In a typical REST architecture, only four HTTP requests are used to manipulate the resources on a server:

a. GET is used to obtain a representation of the resource

b. PUT creates or updates a resource

c. DELETE gets rid of the resource

d. POST appends data to the resource or requests that the server create a subordinate resource

One of the standard RESTful restrictions on use of these HTTP requests is that GET is a “safe” request in the sense that it doesn’t change the server resource state. This means that servers can be probed without upsetting the state of the server itself. Additionally, PUT, and DELETE are idempotent meaning that if the request is sent twice the server is in the same state as if it was sent once. This is important in the case when a response is not received and it is not clear whether the server actually received the command. An idempotent command can be sent again without undesirable side effects. The POST command is neither safe nor idempotent however the Post Once Exactly (POE) approach can be used to make it idempotent.

Another of the aspects of the REST architecture is the concept of stateless communication. This means that each client request must contain all information necessary for the server to evaluate that request, thus keeping the session state is entirely the responsibility of the client. This simplifies the server implementation because it does not have to track the client state from one request to the next. It also simplifies the client implementation since the client does not have to worry about server expectations of its behavior from one request to the next. Because each server does not know the session state of the overall system configuration, fault recovery requires cycling fewer modules in order to restore that overall state.

4.1.3 Discovery

To meet the plug-and-play objectives of Section **Error! Reference source not found.**, three TCP/IP based discovery technologies were assessed; VXI-11[9], UPnP[10], and Zeroconf[11]. Zeroconf was found to be the simplest technology and it is gaining acceptance within the test and measurement industry (LXI and LabView are two examples).

Zeroconf is a combination of three technologies: link-local addressing, Multicast DNS (mDNS) and DNS Service Discovery (DNS-SD).

Link-local addressing allows a network device to make up an IP address for itself in the absence of a DHCP server. This capability is not significant for the Automation Hooks objective as we would expect a DHCP server to be functional in the test lab. In the case where it is not, however, link-local addressing would still find a usable IP address.

Multicast DNS allows computers and devices to negotiate their own locally unique hostnames so that they can be referred to by name without adding hostnames to a DNS server or in the absence of a DNS server.

DNS Service Discovery (DNS-SD) is a service discovery method that works on small networks (perhaps with no infrastructure) with mDNS and on large networks using wide-area DNS Discovery. It allows a DNS-SD client to discover services based on service name and type and then map those services to IP addresses and port numbers.

4.1.4 Extensible Markup Language (XML)

Use of Extensible Markup Language (XML) as a communication and data storage format provides language and operating system independence. It is increasingly being used as a data format and/or a data description language.

Because XML is an open standard, its use as a data representation isolates the components from changes in software versions on other components in the system. This reduces maintenance costs and in the long term will allow legacy equipment to remain functional within a given configuration. Because XML is designed to be easily program-readable, libraries and other tools for development of software utilities to access XML documents are ubiquitous so the data can be transformed for various analysis or display purposes.

A data archive format based on XML has longer shelf life than implementation-specific formats such as database files.

Through the use of schema, XML also provides a standard method (with available tools) for describing document structure and data types. A schema can be used for documentation, validation, data binding, and guided editing.

[9] http://www.vxibus.org

[10] http://www.upnp.org

[11] http://www.zeroconf.org

The main disadvantage of XML as a communication and data archive format is that bandwidth and file sizes are not optimized. The proposed architecture addresses this disadvantage through use of existing HTTP capabilities such as

a. open standards-based compression algorithms like GNU zip (gzip) to reduce document size[12].

b. Conditional GET requests and proxy caching to avoid unnecessary document transfers

4.1.5 Automatic Test Markup Language (ATML)

Automatic Test Markup Language (ATML) is an XML language defined using an interrelated collection of specifications and associated W3C schemas. Its purpose is to define a standard exchange medium for sharing information between components of automatic test systems. The ultimate automation goal is to write test requirements, have the test requirements automatically executed on a test rig, and have an artificial intelligence program analyze the test results, including for diagnostic purposes.

The original mREST prototypes were built upon ATML but it was determined that a better design would be to allow the mREST protocol to support any number of XML-based schemas. This allows each client in the system to communicate in their preferred XML language rather than forcing every server to adhere to one schema. The mREST protocol has a default XML schema that it uses to communicate but a client could provide an XML transformation (in the form of an XSLT document) from mREST to any other XML language.

Since ATML is quickly gaining widespread use in the automated testing community, mREST servers will typically provide ATML support in the interface, although this is not a hard requirement.

4.1.6 Open Source

It is worth mentioning that the fundamental technologies presented in this report, such as HTTP, XML, REST and Zeroconf have widespread support within the open source community.

The open source community emerged towards the end of the 1990's with the rise in popularity of Netscape Navigator and the Linux operating system. Open source software is software that is distributed according a set of criteria[13] that includes:

a. Source code must be made easily available

b. Derived works must be allowed

c. Redistribution is not limited

[12] http://www.gzip.org

[13] http://opensource.org/docs/osd

d. Use is not restricted by license to specific technologies, fields of endeavor, or as part of specific products

Multiple open source implementations of REST, for example, are available for all major operating systems, most computer languages, and most major computing platforms. These implementations also tend to be leaner and more efficient and defect-free than many mainstream commercial software packages.

4.2 System Components

4.2.1 Logical System Element (LSE)

Each Logical System Element (LSE) is a server in the mREST architecture and can represent many different types of objects. LSEs have been developed for such disparate items as measuring equipment, power switches, network cameras, computer systems, simulations, graphical models, internet data sources, and LabVIEW virtual instruments.

An LSE is made up of three main components as shown in Figure 6. The "resources" make up the data that is served as part of the mREST interface and is sometimes referred to as the "LSE front end".

The "parameters" are the internal data elements that are necessary for operation of the LSE. The data here is generally stored using a durable mechanism such as a database. How each parameter is exposed in the list of resources is left up to the LSE designer.

The "core" is sometimes called the "LSE back end" and represents the actual object of interest. The core may include internal interfaces to other software or hardware, as necessary. For example, in a network camera LSE, the manufacturer-provided interface to the camera itself is handled as part of the LSE core. The architecture of the LSE core will vary widely for different types of LSEs.

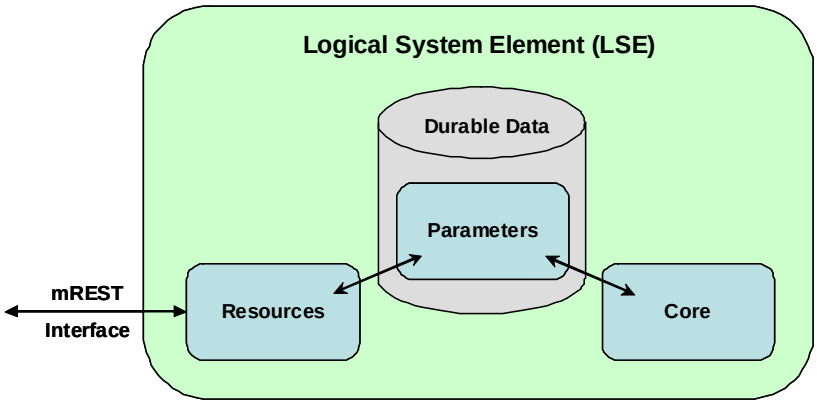


Figure 6: Logical System Element (LSE) Architecture

Regardless of the object involved, the key attribute that makes it an LSE is that it has the common mREST interface. The internal architecture of an LSE is essentially left up to

the developer, although it is generally recommended that the resources, parameters, and core be separated as shown in Figure 6. Eliminating a direct interface between the resources (front end) and the core (back end) keeps the design more compartmentalized and also provides for independent control of each interface. Client requests coming in the front end do not have to make a round trip through the core before a response can be formulated.

To support automatic discovery, each LSE advertises its services through DNS-SD using multicast DNS (mDNS). Use of mDNS is important because it allows the system architecture to more easily scale from large distributed systems to very small systems (at the minimum with all LSEs on the same computer). If the LSE is located on another network such that discovery through multicast DNS will not work then it is referred to as a Remote LSE and several techniques are available for DNS-SD in that situation.

4.2.2 mREST Managers

An mREST manager is effectively any client in an mREST system. The clients are responsible for initiating requests and collecting responses from each LSE (server) in the system. It is typical for an mREST Manager to also act as an LSE so that it can respond to requests from other clients in the system or even respond to internal requests from itself or other copies of itself. A few sample mREST client implementations are described in the following subsections.

4.2.2.1 Test Flow Data Manager (TFDM)

The Test Flow and Data Manager (TFDM) provides overall coordination and control of a test. The TFDM client sends requests to each LSE to gather data and to orchestrate every aspect of the test. When the test is complete, the TFDM documents the test in an ATML format data package for analysis and archival purposes.

The test conductor communicates with the TFDM using a web browser. For this purpose the TFDM incorporates a web server. This allows the test to be conducted from any computer authorized to connect to the TFDM and it mobilizes the test Ccnductor, allowing him to disconnect, move, and reconnect without being disruptive. It also provides a mechanism for allowing observers to monitor the test from their web browsers.

It is expected that there may be different styles of TFDM's developed to accommodate test requirements of specific labs. Currently, a single core TFDM implementation has been used in a variety of test situations by adding a test-specific panel page to the TFDM to provide convenient test-specific controls.

The scope of TFDM responsibilities is primarily intended to be limited to orchestration and data gathering functions. Although a test-specific panel page is convenient for simple test-specific controls, the Standalone Text Exec (STX), described in the next section, is provided for detailed control of test flow. This division of responsibilities allows for more modular implementation of the two types of mREST Managers.

4.2.2.2 Standalone Test Executive (STX)

A certain amount of test-specific software for commanding, monitoring, and performing real-time calculations to support a specific test objective could be incorporated as a custom module within the TFDM software. For situations where a test executive must be tightly interfaced to a test set or where the test executive is to be more portable, a Standalone Test Exec (STX) may be used. This STX may have test-specific connections to the test set or to each LSE. It must also have an LSE interface which will be used by the TFDM to configure, control, and monitor the STX. The STX may have a web services client in which case it may communicate with designated LSEs using the same protocol as the TFDM but with restrictions on which LSE resources it may modify.

4.2.2.3 Facility Manager (FM)

The Facility Manager (FM) client application is used solely to detect LSEs on a network and to make them available for use in a given facility. This client application gives a facility manager control over which assets he/she wishes to make available for a given test or simulation session. The FM application also contains its own LSE interface.

4.2.2.4 Session Manager (SM)

The Session Manager (SM) client application is used to assemble LSEs for a specific session configuration. The session configuration is simply the grouping of LSEs that are used for a given purpose. The group of available LSEs can be automatically discovered on the network by the SM or they can be obtained via the LSE interface on the Facility Manager (FM) application. The SM application also contains its own LSE interface.

4.3 Internal LSE Data Modeling

While not specifically a requirement of this specification, it is generally good practice to model the data within an LSE in a manner consistent with other mREST agents. It is also recommended that LSE designers use common terminology. A discussion of the concept of parameters and resources as well as static and dynamic parameter data is presented in the following subsections.

4.3.1 Parameters and Resources

For the purposes of this document, the word “parameter” has a specific meaning with regard to the modeling of data internal to an LSE. A parameter is an abstraction of the internal pieces of LSE data whereas a resource is the exposure of data in the mREST interface. How the internal parameters are mapped to resources in the mREST interface is one of the main design decisions that the LSE designer must confront. As previously shown in Figure 6, parameters are for “internal” LSE data and resources are for the “external” LSE data.

It might be tempting to think of a parameter as simply a “variable” in the LSE software. However, a parameter can have many attributes which might actually include several different variables in the actual software. For example, a single parameter that stores the

value of a temperature sensor might also include the name, description, type, units, and limit values.

4.3.2 Static and Dynamic Parameter Data

When designing the representations for the lowest level resources, the mREST specification makes an attempt to separate a parameter's dynamic data from its static metadata. Data gathering in an mREST system is usually performed in a somewhat cyclical manner and thus some of the following benefits can be realized by reducing the amount of static data that has to be returned with each response:

1. Reduced verbosity of each representation,
2. A reduced command/response latency between clients and servers,
3. The ability for static metadata to be cached and thus reduce the load on each server

In general, the only "dynamic" portion of a parameter in a system is its value. Exceptions might include confidence intervals or tolerances which are situational metadata. The name of the parameter is also included in the dynamic data so that the value can be referenced properly. Everything else can be considered static metadata unless for some reason it changes dynamically (Table 3).

Table 3: Static and Dynamic Parameter Data

Dynamic Parameter Data	Static Parameter Data
name	description
value	type
	units
	limit values (max, min)
	expected values
	user interface characteristics
	any other metadata (extensible schema)

To increase flexibility and to serve various types of clients, a single resource representation that includes both the static and dynamic data should also be provided. So, in general, there could be at least 3 resource representations for each parameter:

1. Dynamic data
2. Static metadata
3. Complete data set

The mREST specification accomplishes this by providing a mechanism for declaring all three resources for each parameter and then using ATOM links to relate the three resources. A few of the advantages to this approach are:

1. This provides greater flexibility since the LSE designer can still choose to model data in a traditional sense (1 resource for each parameter) while providing optional finer grained controls for dynamic data without the overhead of potentially large payloads of metadata or ignored parameters.
2. Full caching can be employed with the static resources thus reducing the load on the LSE server
3. Resource updates from the client are simpler since a parameter's value can be modified without maintaining the metadata. Likewise, a parameter's metadata can be updated without affecting its current value.

For example, consider a simple integer parameter in an LSE that has the attributes listed in Table 4.

Table 4: Sample LSE Parameter

Attribute	Value	Type
Name	pint	dyndata
Value	6	
Description	This is a sample integer parameter	metadata
Type	Integer	
Units	V	
Max Value	10	
Min Value	0	
Expected Value	5	

An LSE designer provides three separate resources that map to this parameter:

1. /mrest/test/pint – A resource with all of the parameter data
2. /mrest/test/pint/metadata – A resource with just the static parameter data
3. /mrest/test/pint/dyndata – A resource with just the dynamic parameter data

The representation for each resource would include two ATOM links to the other resources that are mapped to this parameter. Thus, if a client retrieved just the dynamic data, it would be provided a reference to where it could also retrieve the static data or the complete data set.

The LSE designer would also set the appropriate cache expiration flags so that the static data would only have to be provided once for each client, thus eliminating the processing associated with responding to multiple client requests for data that is not changing.

5 mREST Interface Definition

5.1 Resource Types

Rather than enforce each mREST server to have a specific list of resources, it is more general to define a set of resource "types" that encompass all of the possible resources that an LSE designer might wish to employ. Resources of a given type will exhibit similar behavior even if their content is different. A list of all the allowable resources types is provided in Table 5 with more detailed descriptions provided in the following subsections.

Table 5: mREST Resource Types

Resource Type	Description
Document	A document that the LSE manages without making any attempts to process or convert to another format; designed for relatively static resources
Component	The lowest level piece of data in the LSE resource tree
Collection	A collection of other resources; the representations for each of the resources in the collection are provided in the response
Catalog	A catalog listing of other resources; similar to a collection except that the representations for the resource in the catalog are not provided in the response
Controller	A controller resource performs an action on other resources or data in the LSE; sometimes also called a callback resource
Datalog	A datalog resource is used to present data that has been recorded locally on the LSE.
Redirect	A redirect resource forwards all requests to another resource on the LSE.

5.1.1 Document Resources

A document resource is a resource that can be of any format. The mREST server does not attempt to translate or otherwise process the content of a document resource. Document resources are best used for the storage of documents that are updated and retrieved at a relatively low frequency. The command set for a document resource is shown in Table 6. Documents that must not be corrupted may be read-only, the server is only required to support "GET".

Table 6: Methods for mREST Document Resources

Method	Data Payload	Resource Behavior
GET	--	Return document resource

PUT	Full document representation	Update existing document resource
POST	Full document representation	Create new document resource
DELETE	--	Delete existing document resource

5.1.2 Component Resources

A component resource is the lowest-level piece of data in the LSE resource tree. Typically, a single parameter in the LSE would be mapped to a single component resource. However, a component resource could also consist of a list of parameters if the LSE designer does not wish to map each parameter to a single resource. The drawback to this approach is that there would be no way of accessing just one of the parameters in the list via the resource tree. The recommended approach is to map each parameter to a single resource and then use a collection resource to provide a grouping of several parameters. The command set for a component resource is shown in Table 7.

Table 7: Methods for mREST Component Resources

Method	Data Payload	Resource Behavior
GET	--	Return resource representation
PUT	Component resource representation	Update existing component resource
POST	Component resource representation	Create new component resource
DELETE	--	Delete existing component resource

5.1.3 Collection Resources

A collection resource is a grouping of other resources in the LSE resource tree. Typically a collection will include a group of component resources but it is acceptable to include any type of resource in the collection. A collection can even include other collections. When a collection resource representation is requested, the server returns the representations for each of the resources in the collection. The command set for a collection resource is shown in Table 8.

Table 8: Methods for mREST Collection Resources

Method	Data Payload	Resource Behavior
GET	--	Return representations for all resources in collection
PUT	Collection resource representation	Update each resource in collection
POST	Listing of resources in collection	Create new or modify existing collection resource
DELETE	--	Delete existing collection resource

5.1.4 Catalog Resources

A catalog resource is a listing of other resources in the LSE resource tree. A catalog is similar to a collection except that the representations for the resource in the catalog are not provided in the response. The command set for a collection resource is shown in Table 9.

Table 9: Methods for mREST Catalog Resources

Method	Data Payload	Resource Behavior
GET	--	Return listing of resources in catalog
PUT	--	--
POST	Listing of resources in catalog	Create new or modify existing catalog resource
DELETE	--	Delete existing catalog resource

5.1.5 Controller Resources

A controller resource shall be used to execute a callback function which performs an action on other resources or data in the LSE. This action may happen immediately or be delayed until a scheduled time or for a period of time. Execution of the controller may also take a significant amount of time. A separate set of children resources to record the execution input, status, and process id (PID) shall be created each time a controller is executed (Figure 7). These status resources provide a mechanism for the server to return an immediate response while also providing a way for the client to continually check the controller status even after the controller execution has completed.

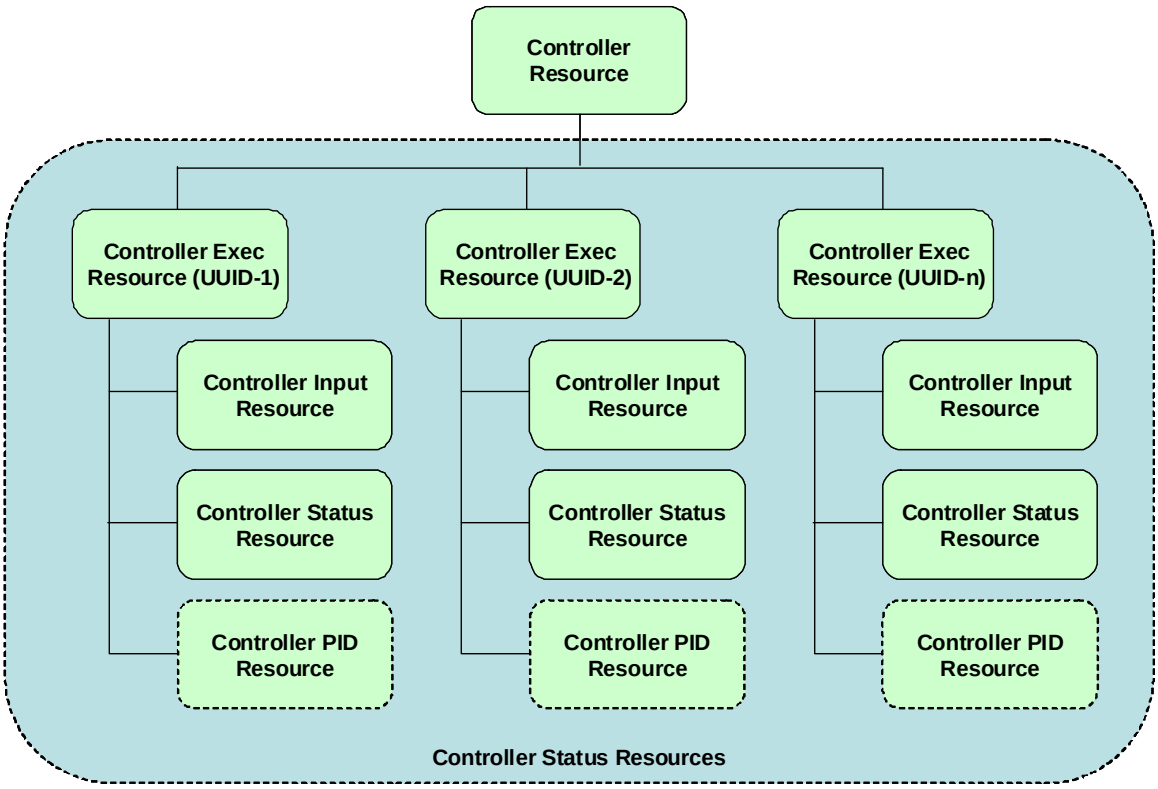


Figure 7: Controller Status Resources

Each time a controller resource is executed, a "controller exec" resource shall be created which contains a universal unique identifier (UUID) referring to that execution instance. A "controller input" resource shall also be created to document the input parameters that were passed to the controller upon execution. The "controller status" resource shall provide status information about the current execution of the controller. The controller input and status resources shall continue to exist on the server until they are deleted by the client or the LSE is reset.

If the controller is in the midst of execution, a "controller PID" resource shall contain the actual process identifier (PID) on the LSE machine. This resource will not exist if the callback has not begun execution or if its execution has completed. A client can kill the controller process by sending a DELETE command to the "controller PID" resource (or the "controller exec" parent resource).

A controller resource shall be executed by sending a POST command with the appropriate inputs for the controller. The server shall create the associated controller children resources and respond with the "controller exec" resource representation. PUT and DELETE are not required to be supported for the main controller resource. The command set for the main controller resource is shown in Table 10.

Table 10: Methods for mREST Controller Resources

Method	Data Payload	Resource Behavior
GET	--	Return all status resources for this controller
PUT	--	--
POST	Controller input parameters	Execute controller resource and create status resources
DELETE	--	--

A GET to any of the controller status resources shall return the resource representation for that resource. A DELETE to any of the status resources shall delete that resource and all of its children from the resource tree. If the controller PID resource is deleted, the associated process on the LSE hardware shall be killed and the status resource updated to reflect the fact that the controller was killed by client request. PUT and POST methods are not required to be supported for controller status resources. The command set for the controller exec, input, and status resources is shown in Table 11.

Table 11: Methods for mREST Controller Exec, Input, Status, and PID Resources

Method	Data Payload	Resource Behavior
GET	--	Return respective status resource representation
PUT	--	--
POST	--	--
DELETE	--	Delete resource and all associated children

5.1.6 Datalog Resources

A datalog resource shall be used to present data that has been recorded locally on the LSE. The datalog resource is essentially a list of parameters with each record tagged with a timestamp for when the data was logged. An ATOM link to the controller exec resource that created the datalog shall be provided as part of the resource. An ATOM link for each resource that is included in the datalog shall also be provided.

The creation and updating of a datalog resource is handled internally to the LSE (see Section **Error! Reference source not found.**) so the only methods supported for this resource are GET and DELETE as summarized in Table 12.

Table 12: Methods for mREST Datalog Resources

Method	Data Payload	Resource Behavior
GET	--	Return resource representation
PUT	--	--
POST	--	--

DELETE	--	Remove datalog resource

5.1.7 Redirect Resources

A redirect resource shall be used to map one resource to another resource on the LSE. All requests received at a redirect resource are automatically forwarded to the target resource. The command set for a redirect resource is shown in Table 13.

Table 13: Methods for mREST Redirect Resources

Method	Data Payload	Resource Behavior
GET	--	Return representation of target resource
PUT	--	Update existing target resource
POST	Redirection target resource	Create new or modify existing redirect resource
DELETE	--	Delete existing redirect resource

5.2 Standardized Controller Resources

5.2.1 Contents collection?

5.2.2 Synchronization (/mrest/sync)

System time on the server host. Generally provided as-is, a client can determine whether the server is tightly synchronized. If not, the condition may be reported to an operator and the client may compensate execution start times in schedule requests to controller resources. This resource shall be provided if schedulable controller resources are implemented.

5.2.3 Data Logging (/mrest/controllers/logdata)

Datalogging on an LSE shall be managed via a controller resource (Section 5.1.5). A datalog is initiated by sending a POST command to the datalog resource at */mrest/controllers/logdata* with the appropriate inputs. Upon execution, the datalog controller shall provide the standard controller response as well as an ATOM link to a datalog resource (Section 5.1.6) which contains the resulting data records.

The only input that shall be required for a datalog controller is a list of the resources to be recorded. This will result in the LSE recording a single snapshot of the provided resources.

The option to provide a starttime/offset, endtime/duration, and/or an interval shall also be supported. If a start time or offset is provided, the LSE will wait the specified amount of time before datalogging is initiated. If an endtime or duration is provided, the LSE will stop logging data at the appropriate time. If an interval is provided, the LSE will create a

new record at each interval. If an interval is not provided, the LSE will only log a single snapshot and thus any endtime or duration input would be ignored.

A list of the input parameters for a datalog controller is provided in Table 14.

Table 14: Inputs for Datalog Controller Resource

Input	XML Element	Description
Start Time	<starttime>	XML formatted date/time when datalogging should begin An offset can be provided in lieu of a start time Datalogging shall begin immediately if a start time or offset is not provided
Offset	<offset>	The elapsed time to wait before datalogging should begin A start time can be provided in lieu of an end time Datalogging shall begin immediately if a start time or offset is not provided
End Time	<endtime>	XML formatted date/time when datalogging should end A duration can be provided in lieu of an end time Only a single record shall be recorded if an end time or duration is not provided
Duration	<duration>	The duration in seconds of datalogging An end time can be provided in lieu of a duration Only a single record shall be recorded if an end time or duration is not provided
Interval	<interval>	The interval in seconds between each datalog record Only a single record shall be recorded if an interval is not provided
Resources	<resource>	The resource path to be logged Any number of resource elements can be listed but at least one must be provided in order for the datalog to contain any records

In summary, to command the LSE to record a single snapshot of data, the client would provide the following inputs:

1. A list of the resources to be logged
2. An optional start time or offset.

To command the LSE to initiate cyclic data logging, the client would provide the following inputs:

1. A list of the resources to be logged
2. An interval for the cyclic data logging
3. An optional start time or offset
4. An optional end time or duration

The XML document in Table 15 is an example of the XML input for a datalog controller. This example lists four resources to include in the datalog which will occur cyclically for 5 seconds at a 1 second interval starting at the specified date and time. This will result in the output of five records in the associated datalog resource.

Table 15: Sample XML Input for Datalog Controllers

XML Document

```
<?xml version='1.0' encoding='UTF-8'?>
<mrest:controller xmlns:atom='http://www.w3.org/2005/Atom'
xmlns:mrest='http://www.mrest.org/2012/mREST' >
<mrest:controller_input>
  <mrest:starttime>2012-05-15T16:31:35</mrest:starttime>
  <mrest:duration>5.0</mrest:duration>
  <mrest:interval>1.0</mrest:interval>
  <mrest:resources>
   <mrest:resource>/mrest/test/pint/dyndata</mrest:resource>
   <mrest:resource>/mrest/test/pint/dyndata</mrest:resource>
   <mrest:resource>/mrest/test/pfloat/dyndata</mrest:resource>
   <mrest:resource>/mrest/test/pstring/dyndata</mrest:resource>
  </mrest:resources>
 </mrest:controller_input>
</mrest:controller>
```

If the “starttime” element was not included in the input, then data logging would execute immediately. If the “interval” element was not included, then the “duration” element would be ignored and only a single snapshot of the listed resources would be recorded.

The complete XML schema for the datalog controller inputs is provided in Table 16.

Table 16: XML Schema for Datalog Controller Inputs

XSD Document

```
<?xml version='1.0' encoding='UTF-8'?>
<xs:schema targetNamespace="http://www.mrest.org/2012/mREST"
xmlns:xs='http://www.w3.org/2001/XMLSchema' xmlns:atom='http://www.w3.org/2005/Atom'
xmlns:mrest='http://www.mrest.org/2012/mREST' elementFormDefault="qualified">
<xs:element name="controller">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="controller_input" minOccurs="1" maxOccurs="1">
    <xs:complexType>
     <xs:all>
      <xs:element name="starttime" type="xs:dateTime" minOccurs="0"/>
      <xs:element name="endtime" type="xs:dateTime" minOccurs="0"/>
      <xs:element name="offset" type="xs:decimal" minOccurs="0"/>
      <xs:element name="duration" type="xs:decimal" minOccurs="0"/>
      <xs:element name="interval" type="xs:decimal" minOccurs="0"/>
      <xs:element name="resources" minOccurs="1" maxOccurs="1">
       <xs:complexType>
```

```
        <xs:sequence>
         <xs:element name="resource" type="xs:string" minOccurs="1"
maxOccurs="unbounded"/>
        </xs:sequence>
       </xs:complexType>
      </xs:element>
     </xs:all>
    </xs:complexType>
   </xs:element>
  </xs:sequence>
 </xs:complexType>
</xs:element>
</xs:schema>
```

5.3 mREST Uniform Interface

5.3.1 HTTP Requests

XXX

5.3.2 HTTP Responses

XXX

5.3.3 HTTP Headers

XXX

5.4 mREST Server Resources

5.4.1 Identification

XXX

5.4.2 Data Logging

XXX

5.4.3 Application Specific Resources

XXX

5.4.3.1 Test Environment

XXX

5.4.3.2 Simulation Environment

XXX

5.5 mREST Resource Representations

5.5.1 Resource Representation Formats

Resources in an mREST server shall provide a default representation using the mREST XML media type (`application/vnd.mrest+xml`) as defined in Appendix B.1.

The generic media type `application/xml` shall be mapped to the mREST XML media type (`application/vnd.mrest+xml`) representation format.

Additional resource representations shall be defined using XSLT files available under the resource `/mrest/xslt`. Each XSLT file provides the mapping between the default mREST XML representation format and the additional representation format.

All XSLT files that are used to create new resource representation formats shall use a naming convention based on the media type (`application/vnd.<xslt_filename>`). For example, an XSLT file that transforms the default mREST XML format to the mREST JSON format would be called "`mrest+json.xslt`" and be available at the resource `/mrest/xslt/mrest+json`.

A GET to the resource `/mrest/xslt` shall return a list of the media type formats supported by the mREST server.

mREST servers shall provide the capability for clients to define custom representation formats by posting an XSLT file to the resource `/mrest/xslt`.

5.5.2 Media Type Negotiation

An mREST server shall choose the representation format from a list of accepted media types and an associated query parameter in the HTTP `Accept` header field provided by an mREST client.

The media type query parameter in the HTTP Accept header field provided by an mREST client shall have a decimal range of 0.0 to 1.0 where 1.0 is used to indicate the preferred representation format and 0.0 is used to indicate formats which the client cannot process.

For example, a client who prefers to communicate with the mREST server using ATML (`application/vnd.atml+xml`) but could also communicate using the MREST XML format (`application/vnd.mrest+xml`) would provide an HTTP Accept header of the form:

```
Accept: application/atml+xml;q=1.0, application/mrest+xml;q=0.8
```

If for some reason the mREST server was unable to provide resource representations in the ATML format then it would fall back to the mREST XML format.

5.6 mREST Status Requests

XXX

5.7 mREST Configuration Update Requests

XXX

5.8 mREST Logical System Element Discovery

XXX

5.9 mREST Security

mRest security could include facets of HTTPS, machine-machine trust (certificates), user/operator trust (user authentication). The security architecture has to provide an automation-friendly environment, as the definition will allow one user to control a large collection of software modules, but that's all defeated if he/she has to manually manage and intervene with a large number of passwords. Further, mREST does not specify addresses or port numbers, it delegates management of the low-level configuration to machines. We can have firewalls, but to be general we also must support some distributed applications where some of the system elements are outside the firewall on an open network.

We should consider three levels of access:

1) access denied. Server is discoverable but services not available

2) read-only. Server state or sensor data is available but control is denied

3) read-write. Server can be controlled or updated

6 mREST Manager Requirements

6.1 LSE Interface

6.1.1 Uniform Interface

XXX

6.1.2 Locations of LSE Resources

XXX

6.1.3 Data Representations of Resources

XXX

6.1.4 Caching

XXX

6.1.5 XML Validation

XXX

6.1.6 Stateless Communication

XXX

6.1.7 CallBack

XXX

6.1.8 Optimization

XXX

6.2 Operator Interface

XXX

6.3 Initialization of the System and Session Configuration

XXX

6.4 Coordination of Session Run Execution

XXX

6.5 Collection and Display of Status Data

XXX

6.5.1 Database Verification

XXX

6.5.2 Status Log Request Data

XXX

6.6 Collection of Session Manager Notes

XXX

6.7 Organization of System Session Results

XXX

7 Example Implementations

7.1 Test Flow Data Manager (TFDM)

XXX

7.2 Logical Test Element (LTE)

XXX

7.3 Standalone Test Executive (STX)

XXX

Appendix A mREST Requirements Matrix

sasdf

Table 17: mREST Requirements Matrix

#	Requirement	Reference
1.		
2.		
3.		
4.		
	5.5 Resource Representations	
5.	Resources in an mREST server shall provide a default representation using the mREST XML media type (application/vnd.mrest+xml).	5.5.1
6.	The generic media type application/xml shall be mapped to the mREST XML media type (application/vnd.mrest+xml) representation format.	5.5.1
7.	Additional resource representations shall be defined using XSLT files available under the resource /mrest/xslt.	5.5.1
8.	All XSLT files that are used to create new resource representation formats shall use a naming convention based on the media type (application/vnd.<xslt_filename>).	5.5.1
9.	A GET to the resource /mrest/xslt shall return a list of the media type formats supported by the mREST server.	5.5.1
10.	mREST servers shall provide the capability for clients to define custom representation formats by posting an XSLT file to the resource /mrest/xslt.	5.5.1
11.	An mREST server shall choose the representation format from a list of accepted media types and an associated query parameter in the HTTP Accept header field provided by an mREST client	5.5.2
12.	The media type query parameter in the HTTP Accept header field provided by an mREST client shall have a decimal range of 0.0 to 1.0 where 1.0 is used to indicate the preferred representation format and 0.0 is used to indicate formats which the client cannot process.	5.5.2
13.		
14.		
15.		
16.		
17.		
18.		
19.		

Appendix B mREST Resource Representations

XXX

B.1 mREST XML (application/vnd.mrest+xml)

XXX

B.2 mREST JSON (application/vnd.mrest+json)

XXX

B.3 ATML (application/vnd.atml+xml)

XXX

Appendix C mREST Resource Types

XXX

C.1 mREST XML (application/vnd.mrest+xml)

XXX

C.2 mREST JSON (application/vnd.mrest+json)

XXX

C.3 ATML (application/vnd.atml+xml)

XXX

Table 18: mREST Resource Types

Method	Data Payload	Resource Behavior
Document Resources		
GET	--	Return document resource
PUT	Full document representation	Update existing document resource
POST	Full document representation	Create new document resource
DELETE	--	Delete existing document resource
Component Resources		
GET	--	Return resource representation
PUT	Component resource representation	Update existing component resource
POST	Component resource representation	Create new component resource
DELETE	--	Delete existing component resource
Collection Resources		
GET	--	Return representations for all resources in collection
PUT	Collection resource representation	Update each resource in collection
POST	Listing of resources in collection	Create new or modify existing collection resource
DELETE	--	Delete existing collection resource
Catalog Resources		
GET	--	Return listing of resources in catalog
PUT	--	--
POST	Listing of resources in catalog	Create new or modify existing catalog resource
DELETE	--	Delete existing catalog resource
Controller Resources		
GET	--	Return all status resources for this controller
PUT	--	--
POST	Controller input parameters	Execute controller resource and create status resources
DELETE	--	--
Controller Status Resources		
GET	--	Return respective status resource representation
PUT	--	--
POST	--	--
DELETE	--	Delete status resource and all associated children

IEEE P1877
Test Orchestration Interface
Purpose and Objectives

Chatwin Lansdowne

9/8/2012

What I Want to Achieve

- Architectural Objectives
- Assumptions
- Philosophy
- Trade Study Guiding Principles
- Implications for the Interface
- Measures of “Goodness”

 - Can we define a software and data architecture that will **integrate on a macro-scale**…
 - That we can **produce and use on a micro-scale**

Architectural Choices
for a Test Automation Strategy

- How is relevant data collected and labeled
- How is mutual discovery conducted
- How is communication standardized
- How is test flow orchestrated
- How are tasks outside the test flow facilitated
- How can architecture be scalable to size of test

Criteria for a Software Architecture

- **Platform-independent:** *everyone can use their own appropriate operating system, language, and tools*
- **Inexpensive:** *quick to add, easy to learn, simple to test and maintain*
- **Rapid Assembly:** *quick and easy to integrate and troubleshoot*
- **Data Integrity:** *minimal translations, meta-data capture, archive-quality product, restore by write-back, simplified analysis and reporting*
- **Self-Contained:** *the instructions and documentation are in the interface*
- **Open Standards:** *architectural interfaces can be specified by referencing published non-NASA standards*
- **Non-proprietary:** *support multiple COTS vendors for robustness*
- **Open Source:** *supporting user communities are active and tools and chunks are widely available, widely tested, and widely reviewed*
- **Web-based:** *works with the tools you carry in your pocket*
- **Data-Driven:** *the code can be stable, only support-files change*
- **Low-infrastructure:** *stand-alone capable, minimal reliance supporting infrastructure and staff IT experts*
- **Modularity:** *operations can proceed with broken modules*
- **Durability:** *maintenance is not required for legacy bought-off modules on legacy platforms*
- **Retrofit to compiled code:** *sometimes we have to work with what's available…*
- **Convergence:** *a direction observed in aerospace, test, DoD, and consumer products industries and communities*
- **Versatility:** *the more useful it is, the wider it will be implemented*
- **Scalability:** *scale up– or down to one*

Assumptions: Performance I'm Willing to Trade

- **Frequency of Data Collection:** statistically summarized or triggered captures, not streaming, conditions change 1 sec or slower": a "management layer"
- **Test Styles:** *Parametric Tests*-- Change one variable at a time, *Simulation Runs*-- Multivariate, continuous or event-driven flow
- **Connectivity and Support Services:** 10/100/1G/10G Ethernet; multi-user, multi-platform; firewalled or private nets; everything agrees what time it is(?)
- **Data Storage and Data Types:** Data is not a strip-chart flood, but reduced to a figure-of-merit or snapshot near the source. Need for past *vs.* present *vs.* analysis performance; need named configuration save/restore; near-realtime analysis (ratios, differences) and display of collected data
- **Allocation of Responsibility:** Programming uses high-level rapid-development languages and platforms have significant computing power. Modularity allocates reliability, safety, security to the lowest practical layer.

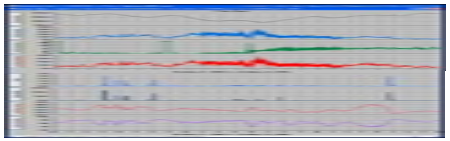
Terrestrial, No DTN

Not flight

What I Didn’t Say

- Security is an elephant in the room
 - Presently relying on traffic security, firewalls, routers, etc.
 - Would like to identify a mechanism that allows expensive instruments to be placed outside the “test ops firewall”, and be managed at arm’s length by any authorized operator controlling the collection through automation.

AHA Prototype Architecture Concept: Data Products



Not just stripcharts

- Analyze before teardown

LTE

LTE

LTE

LTE

PostgreSQL

Post-analysis

Real-time steering

Data

Theory

Engineering Report

Archive product

IEEE

IEEE Standard for Automatic Test Markup Language (ATML) for Exchanging Automatic Test Equipment and Test Information via XML

IEEE Standards Coordinating Committee 20

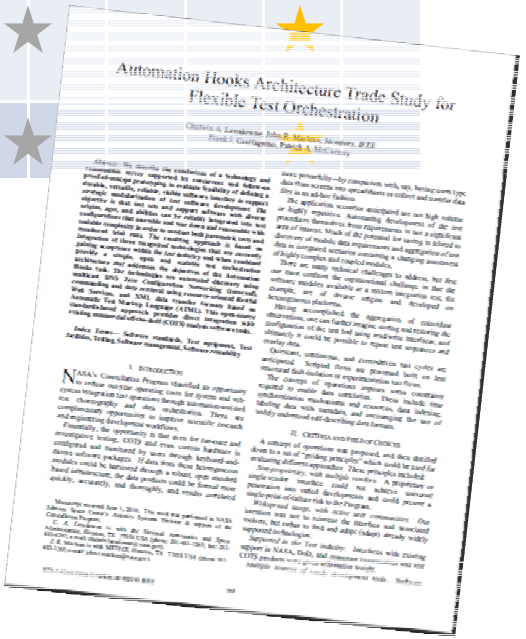
Philosophy of Approach

Test Orchestration and Data Harvest

- Objectives
 - Automate information hand-offs between disciplines
 - Capture archive-quality, repeatable test records
 - Detect emergent behavior in complex systems
 - Reduce development and operations costs
- Principles
 - Do ***not*** restrict tool choices
 - Using documentation in-line makes it accurate and repeatable
 - Data-driven architecture with descriptive interface
 - Simple, general, minimally-restrictive requirements
 - Build on, or make, open-source and open standards

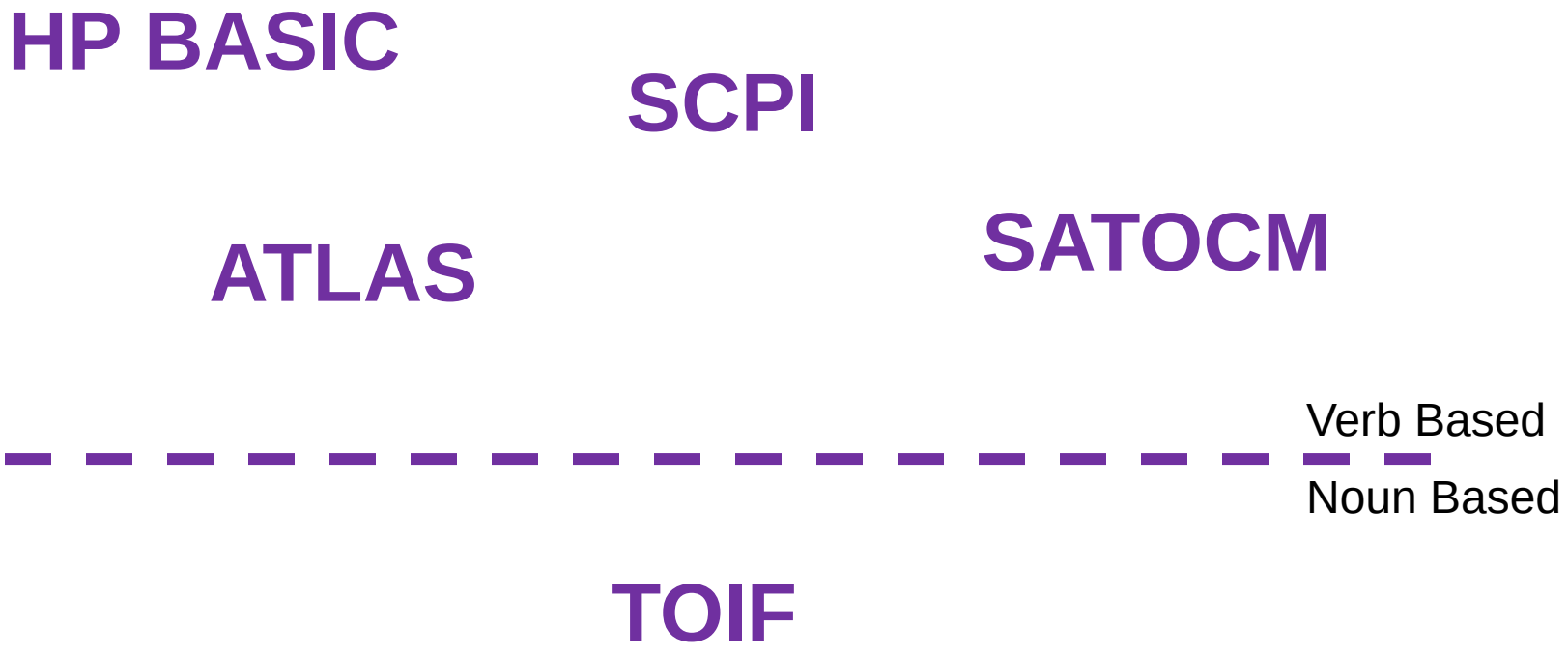
Technology Survey and Trade Study




Automation Hooks Architecture Trade Study for Flexible Test Orchestration

- Surveyed NASA, Test COTS, DoD, and Consumer communities for viable approaches
- Down-selected based on “guiding principles” and prototyping

A Revolutionary New Idea!



- The HTTP command and error-message sets already widely adopted
- Move from Command-Driven to Data-Driven– with REST, the interface is self-describing. Scripting and orchestrating are accomplished by manipulating collections of discoverable “resources.”

Breaking the Information Interface

Client

Test Support: Databases, external support, analysis, reports, user

- Who is using what
- What's connected to what
- Who is doing what
- What is happening and why
- Inventory/Calibration/Location databases
- Data-collecting services
- Data-display services
- Data-analysis services
- Notification services
- Who may use what

Server

Device: Developer describes the "Thing" and the s/w that Controls it

- How to Find it (logical)
- What it is
- Which one it is
- What it knows
- What it does
- How it is configured
- How to configure, calibrate it
- What it is doing/observing now
- What that means
- Who is using it
- Where it is (physical)
- Who may use it

The standard will specify conventional methods, but many of the methods are optional

The Test Results Document

```
<tr:TestResults xmlns:tr="urn:IEEE-1636.1:2011:01:TestResults" xmlns:c="urn:IEEE-1671:2010:Common" xmlns:sc="
urn:IEEE-P1636.99:01:SimicaCommon" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation
="urn:IEEE-1636.1:2011:01:TestResults TestResults.xsd" uuid="{589B634F-10F6-481e-AD22-7115020155BF}">
  <tr:Personnel>
    <tr:SystemOperator ID="jdoe1"/>
  </tr:Personnel>
  <tr:ResultSet ID="_f47ac10b-58cc-4372-a567-0e02b2c3d479" name="BERT.vi" startDateTime="2012-05-30T09:30:10">
    <tr:Outcome value="Aborted"/>
    <tr:Test ID="_6ba7b810-9dad-11d1-80b4-00c04fd430c8" name="my Space-to-Ground Link" startDateTime="
2012-05-30T09:30:10">
      <tr:Parameters>
      <tr:Outcome value="Aborted"/>
      <tr:TestResult ID="_1" name="bits">
      <tr:TestResult ID="_2" name="errors">
      <tr:TestResult ID="_3" name="BER">
      <tr:TestResult ID="_4" name="Time to Acq">
      <tr:TestResult ID="_5" name="Online">
      <tr:TestResult ID="_6" name="Connected">
      <tr:TestResult ID="_7" name="HW Status">
    </tr:Test>
  </tr:ResultSet>
  <tr:TestProgram>
    <c:Definition version="MyVersionOrRevisionNumber">
      <c:Identification designator="My Operator- or Model-supplied identification of instance test-point, configuration
diagram designator, usage, or meaning-- assigned after start-up">
        <c:Version>MyRevisionHistory</c:Version>
        <c:ModelName>MyApplicationName</c:ModelName>
      </c:Identification>
    </c:Definition>
    <c:SerialNumber>MyInstanceUUID</c:SerialNumber>
    <c:ReleaseDate>2011-12-02</c:ReleaseDate>
  </tr:TestProgram>
</tr:TestResults>
```

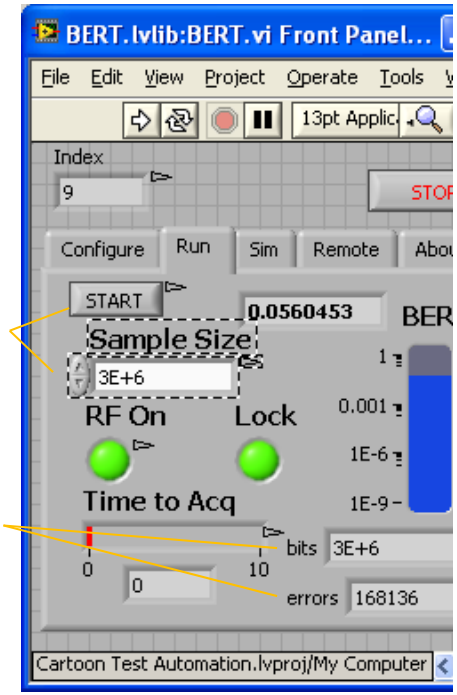
User

Read/write “configuration” variables

Outcome is always “Aborted”

Read-only “status” variables

Software version



- Descriptions could be loaded into tr:TestResults

The Test Description Document

```
<?xml version="1.0" encoding="UTF-8"?>
<td:TestDescription uuid="f47ac10b-58cc-4372-a567-0e02b2c3d479" xmlns:xsi="
http://www.w3.org/2001/XMLSchema-instance"  xmlns:td="
urn:IEEE-1671.1:2009:TestDescription" xmlns:c="urn:IEEE-1671:2010:Common"
xsi:schemaLocation="urn:IEEE-1671.1:2009:TestDescription TestDescription.xsd">
   <td:UUT>
      <td:Description/>
   </td:UUT>
   <td:DetailedTestInformation>
      <td:EntryPoints/>
      <td:Actions>
      <td:TestGroups>
         <td:TestGroup xsi:type="td:TestGroupUnspecifiedOrder" name="none" ID="1">
            <td:Outcomes>
               <td:Outcome ID="1" value="Aborted"/>
            </td:Outcomes>
            <td:ParameterDescriptions>
            <td:TestResultDescriptions>
            <td:ActionReferences>
               <td:ActionReference actionID="0"/>
            </td:ActionReferences>
         </td:TestGroup>
      </td:TestGroups>
   </td:DetailedTestInformation>
</td:TestDescription>
```

- Static metadata is best loaded into tr:TestDescription

Future work: behaviors

Read/write “configuration” variables

Read-only “status” variables

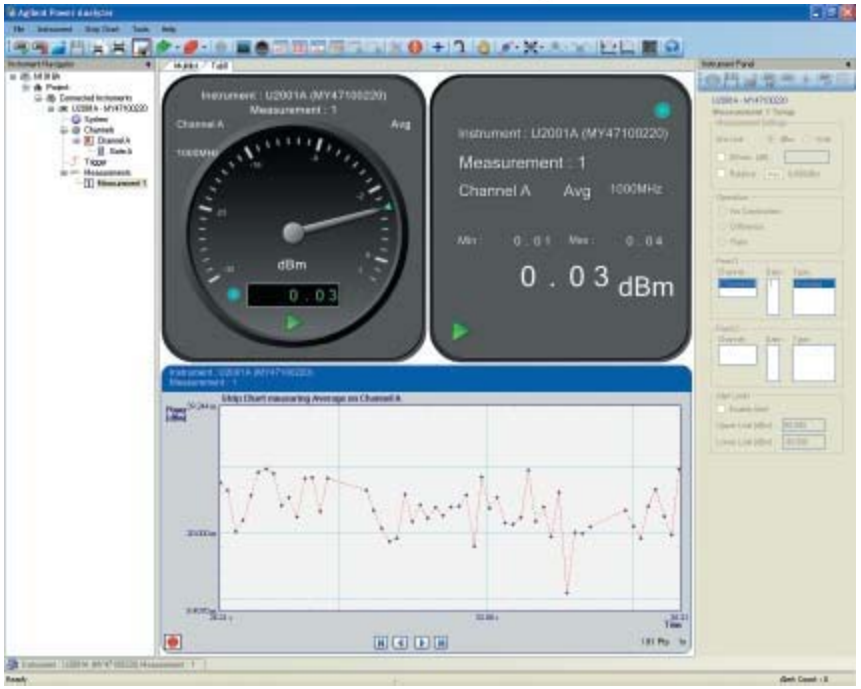
Behavioral Description
Accommodating Alternatives

- Rather that require all software to behave the same, allow developer to describe idiosyncrasies
- Default expected behavior: "PUT" to a resource changes the setting(s) "immediately"
- Some describable alternatives:
 - How long to wait
 - What to check for successful completion: flag, counter, timestamp, measurement…
 - How to write a collection of parameters to the hardware (another PUT after the PUTs)
 - How to clear and restart sticky/accumulative indicators
 - How to abort a measurement
 - How to restart
- Supports configuration RESTORE from SAVEd "GET"

Modern Migration

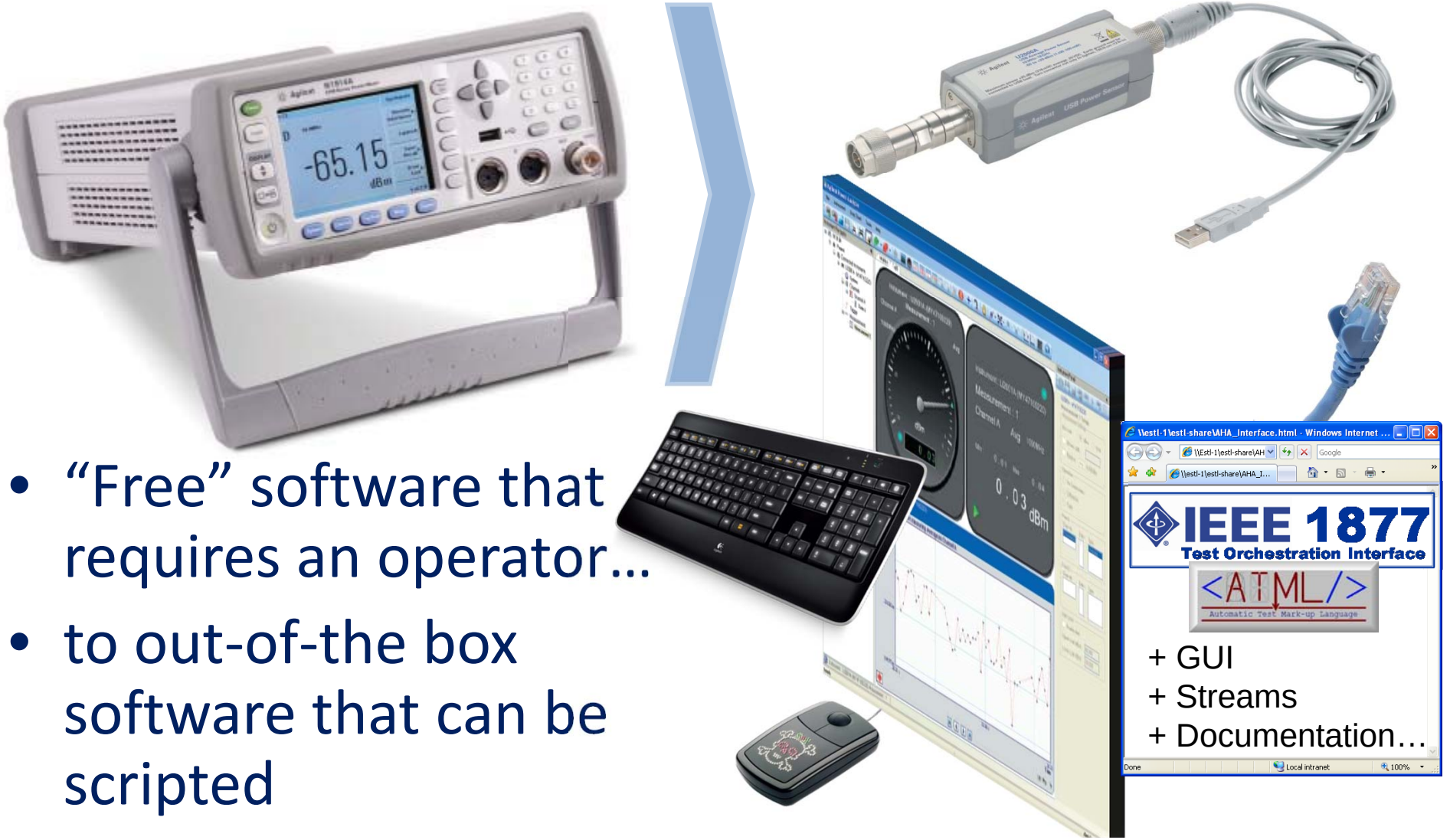






- From dedicated hardware …
- to “headless” USB sensors that come with “free” software.

Modern Migration



- "Free" software that requires an operator…
- to out-of-the box software that can be scripted

You know you're on the right track when…

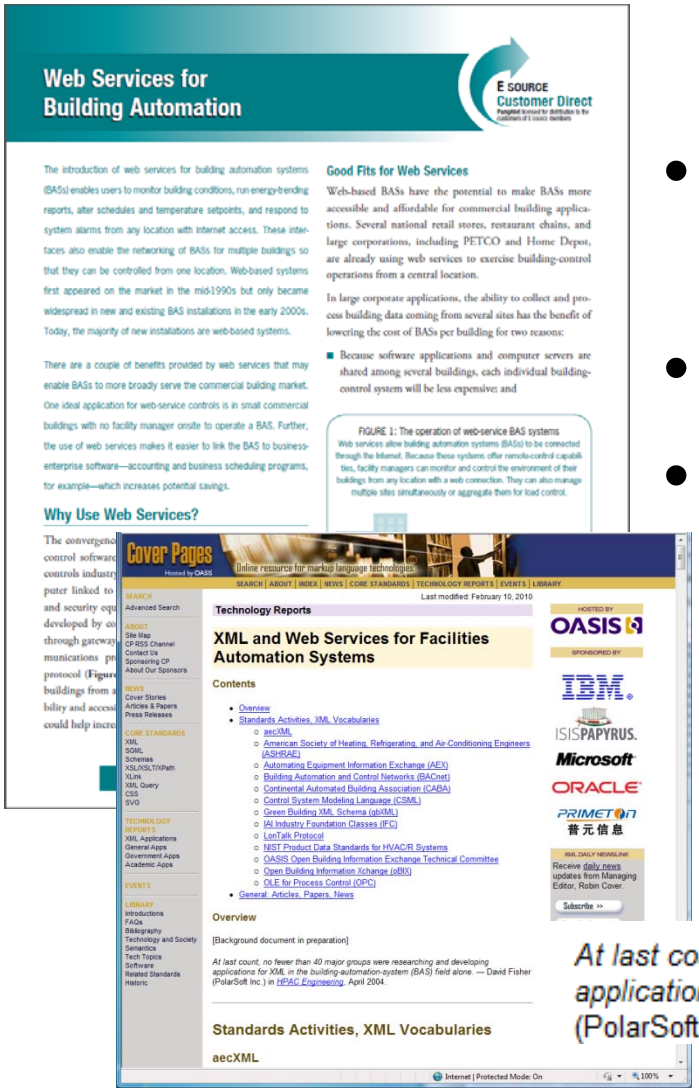
You See

- Interoperability with widely available modern COTS
- Other disciplines actively approaching the problem the same way
- Developers find the complexity empowering not overwhelming

You Don't See

- People managing lists of IP addresses, port numbers, and passwords
- A wordy custom spec, instead of references to other external open standards

An Unexpected Close Ally



Building Automation Systems:

- Interest in web/XML standards is strong
- Security is very important
- Goals: monitoring, diagnostics, prognostics, scheduling, dispatch by expert systems; situationally-aware procedures for technician

At last count, no fewer than 40 major groups were researching and developing applications for XML in the building-automation-system (BAS) field alone. — David Fisher (PolarSoft Inc.) in *HPAC Engineering*, April 2004.

Automation Hooks Architecture API

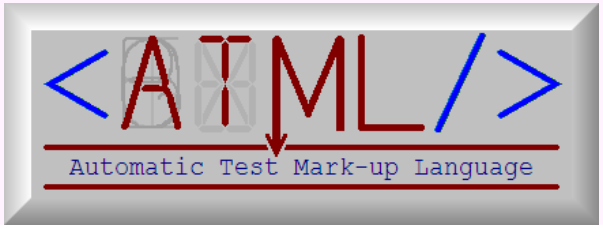
- **Advertised**
 - Automated Discovery: Dynamic “Plug-and-Play”
- **REST Architecture**
 - Two commands: GET and PUT
 - Versatile: co-host support files and hyperlinks– interface definitions, requirements, theory of operation, streaming data, GUI…
- **HTTP**
 - standard messaging, error messages, compression, security, caching

- **Xml**
 - Archive-quality
 - Enables Data-driven software architecture
 - Foundation of artificially intelligent data processing
 - Self-describing message format
 - Create database tables by script
- **hypermedia layout**
 - Insulates against layout changes
 - Coexistence of variations
 - Separate metadata for caching

mREST

- **xml:ATML (IEEE 1671)**
 - standardizes units, arrays, time zone
 - Scope includes signals, instrument capabilities, problem reporting
 - exciting opportunities for COTS tools and radically different engineering work flows

- **Orchestration features**
 - Health and Status Rollup
 - Synchronizing and Scheduling



Testing

BACKUP

Breaking the Interface

(more specific)

Test Support: Databases, external support, analysis, reports, user

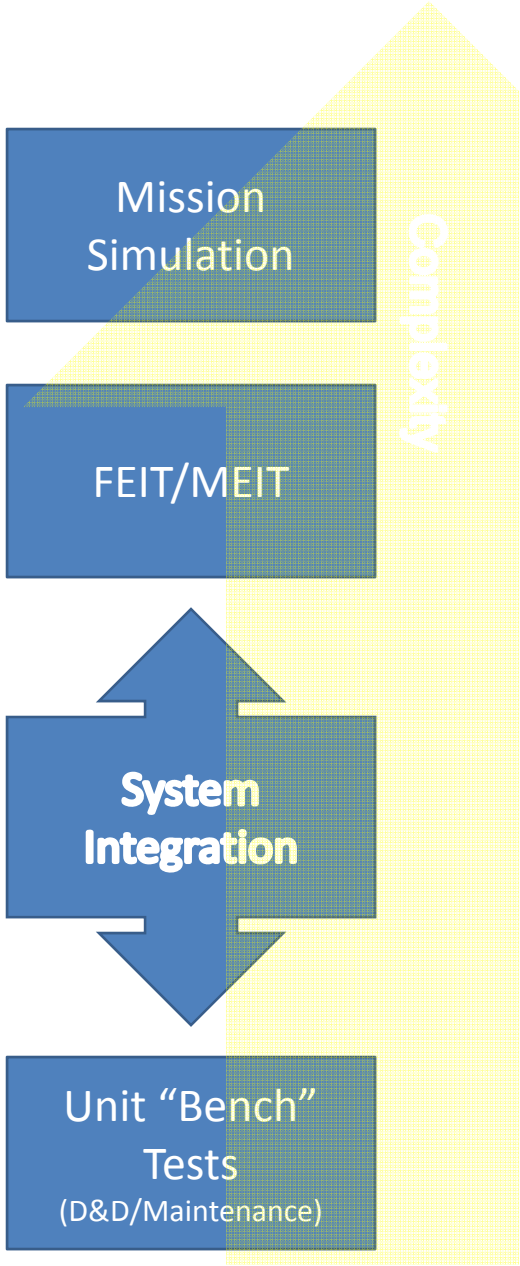
- Who is using what “things”
- Borrowing “things”
- Support services for “things” and Test Execs
 - Database
 - Plotting
 - Configuration Save/Restore
 - Cal Lab, Inventory records
 - Instance management

Device: Developer describes the “Thing” and the s/w that Controls it

- Advertise the information
- Current status/configuration
- What it is
- How to use it
- How to interpret the data
- What the controls do
- Capabilities
- Instance ID
- Who set it up

Automate What?

- Mid to low bandwidth orchestration of both parametric and mission simulation styles of testing
- Coordination and Data Collection from test sets developed by many different Vendors/specialists
- “Run-once” and Evolving Test Configurations, not just permanent testbeds.



Architectural Choices

- Discovery, Data collection, Communication, Scalability
 - Based on open systems technologies developed for WWW
 - Defined standard sets of RESTful resources for data monitoring and control
 - This approach is applicable to many remote or distributed monitoring and control applications
- Orchestration of Test Flow
 - Automatic Test Markup Language provides an IEEE standard communication and data storage language
 - Set of Test flow concepts (next page) was defined to take advantage of these technologies
 - No orchestration command set is required – resource-based instead
- How are tasks outside the test flow facilitated
 - Use of web services provides interoperability between human and software interfaces
 - Test interfaces can be added to existing interactive control panels (Labview) to preserve manual operation capability
 - Test Flow concepts allow flows to branch off for parallel testing or debugging
- How can architecture be scalable to size of test
 - Technologies are lightweight and portable
 - Test elements can be run on a single PC or distributed across a network

Six AHA Test Flow Concepts

- Logical Test Element (LTE)
 - Resource-oriented interface
- Test Flow and Data Manager (TFDM)
 - Discover, overall test flow and data collection
- Standalone Test Exec (STX)
 - Test specific automation/expertise
- Hierarchical Organization of Activities and Data
 - Test Configuration
 - Test Run
 - Data Log Request

Restoring the Viability of NASA's Facilities and Developments

The need for Modern Standards and Practices

- Common tools and Portability of skills
- Agility: Flexibility **and** Speed
 - Fewer standing, dedicated capabilities
 - Reuse/redeployment of assets and people
- Increased quality and detail in Data Products
 - No typos
 - More statistical significance and resolution
 - Ability to locate and interpret "cold" data
 - Analyzing "sets" not "points"

A Scale-to-One Architecture

- Box-Level Development
- Subsystem Integration Test
- System Integration Test
- Multi-Element Integration Test





PostgreSQL



EnterpriseDB
The Enterprise Postgres Company



Orchestrator Development



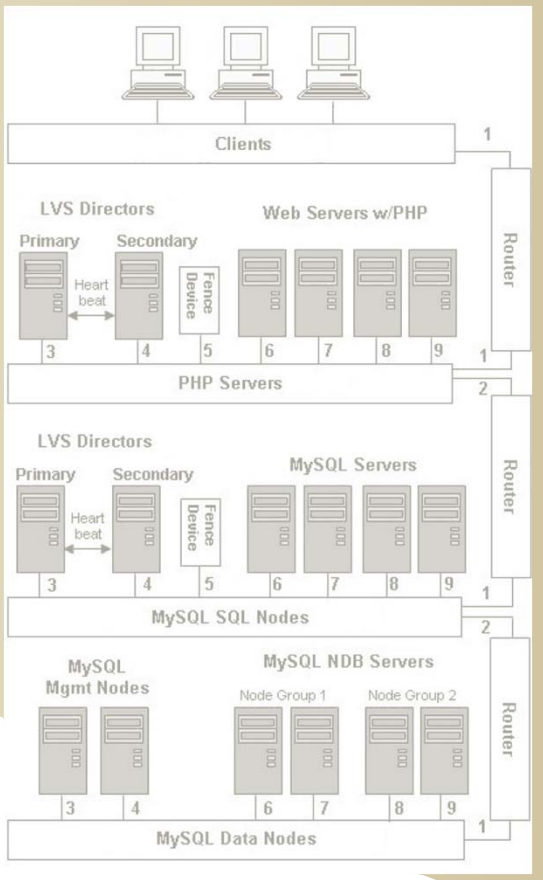
Stackable Orchestrator Development

MySQL Community Server

MySQL Enterprise

Microsoft SQL Server 2008 R2 Express

Microsoft SQL Server 2008 R2 Enterprise



Adding ^Automation without Adding ^Infrastructure



*IT support scales **UP**,* but can IT support scale DOWN?



*IT infrastructure can scale **UP**,* but can IT infrastructure scale DOWN?